PROOF
Present and Future

G. GANIS
CERN / PH-SFT

Technology Seminar, BNL, April 25 2008

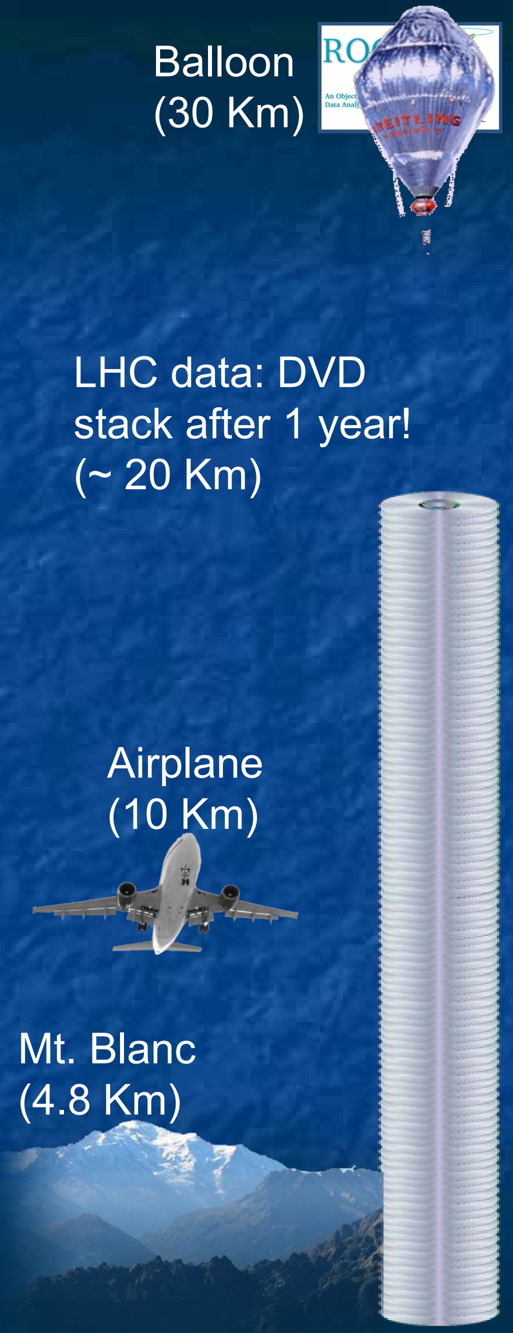
Outline

- Introduction
- Developments for LHC
  - Dataset handling
  - Multi-User management
  - Condor integration
- Other developmnets
  - PROOF-Lite: exploiting multicore
- Summary

LHC Data Challenge

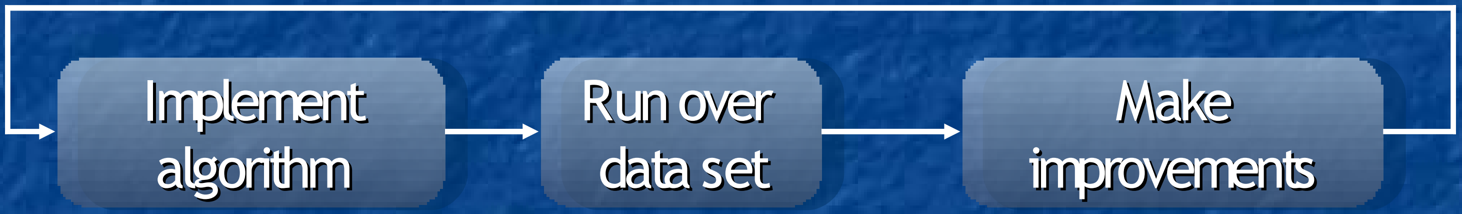
- The LHC generates $40 \cdot 10^6$ collisions / s
- Combined the 4 experiments record:
  - 100 interesting collision per second
  - 1 ÷ 12 MB / collision $\Rightarrow$ 0.1 ÷ 1.2 GB / s
  - ~ 10 PB (10^{16} B) per year (10^{10} collisions / y)
  - LHC data correspond to $20 \cdot 10^6$ DVD's / year!
- Space equivalent to 400.000 large PC disks
  - Computing power ~ 10^5 of today's PC

- Using parallelism is the only way to analyze this amount of data in a reasonable amount of time



HEP Data Analysis

- Typical HEP analysis needs a continuous algorithm refinement cycle



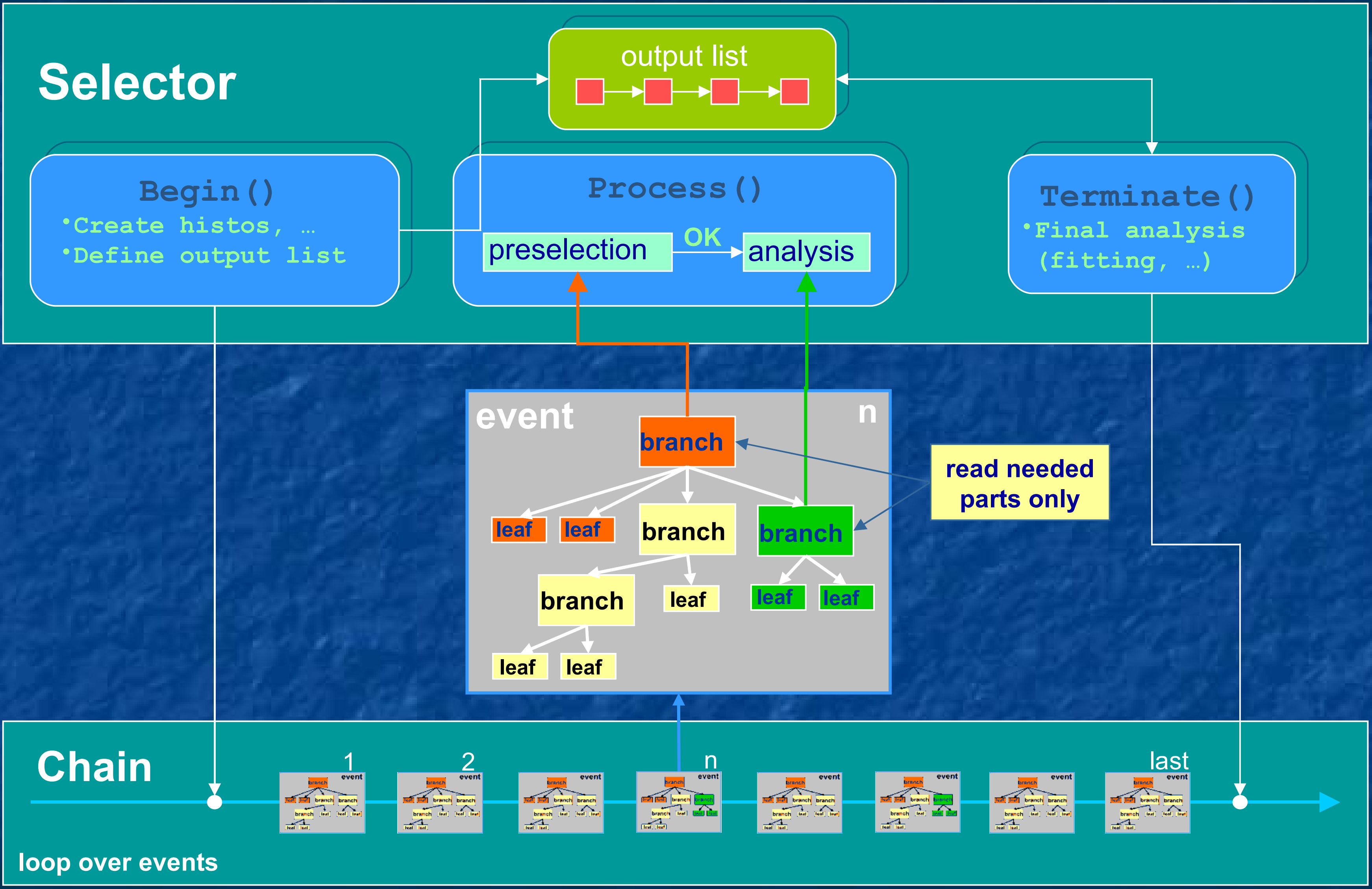
- Typically I/O bound
- Need many disks to get the needed I/O rate
- Need many CPUs for processing
- Need a lot of memory to cache as much as possible

ROOT

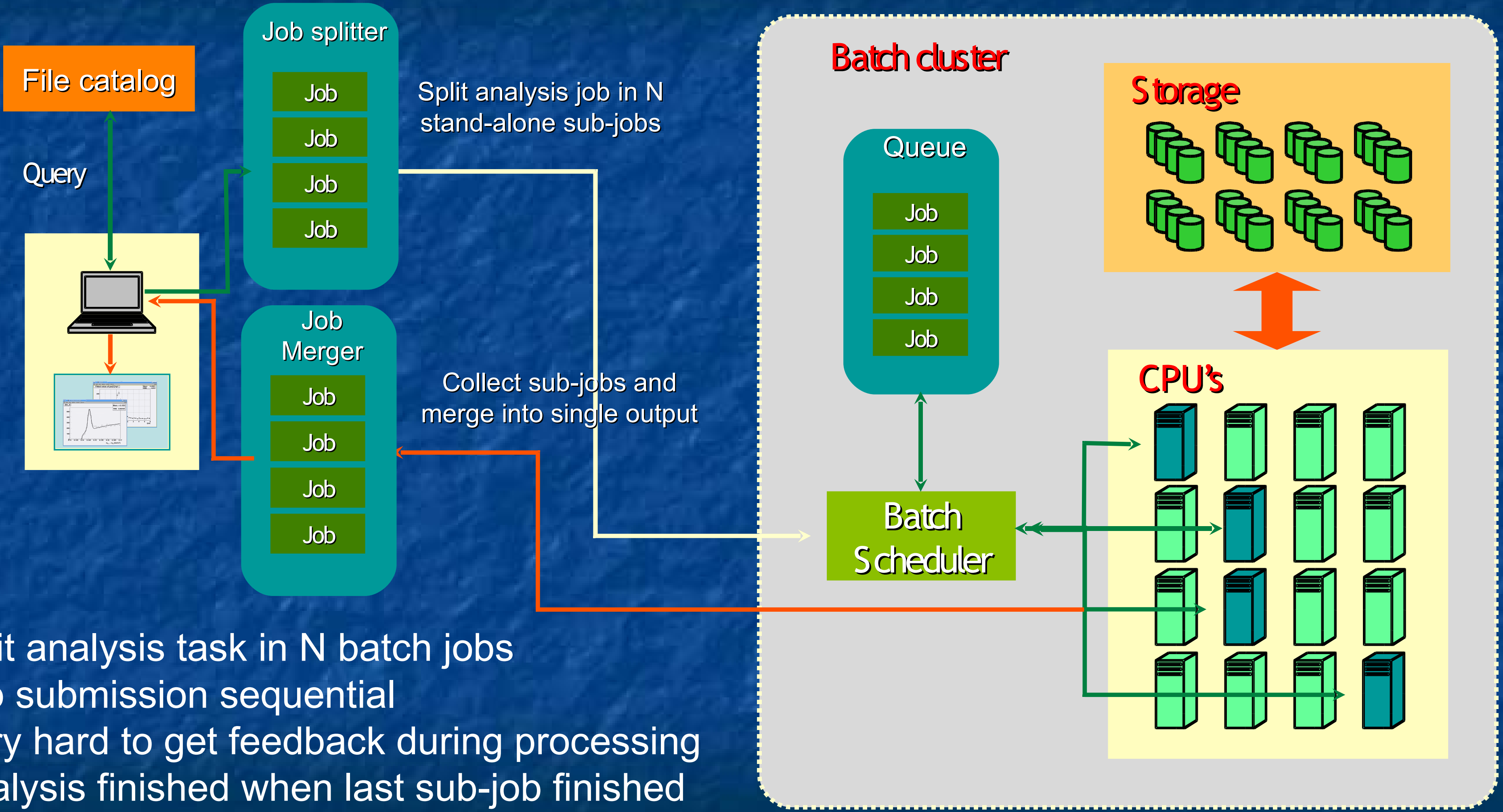
- C++ evolution of the “à la PAW” approach to HEP analysis
- Framework providing tools for
  - Storage optimized for HEP data
  - Visualization (2D, 3D, event display, ...)
  - Statistics, math fucntions, fitting, ...
  - Abstract interfaces (VMC, ...)
- Puts together what was PAW, ZEBRA, CERNLIB and more

- The first and more resource demanding step in a typical HEP analysis is *data reduction*:
  - go through the data and extract the needed info

The ROOT data model: Trees & Selectors



The Traditional Batch Approach



- Split analysis task in N batch jobs
- Job submission sequential
- Very hard to get feedback during processing
- Analysis finished when last sub-job finished

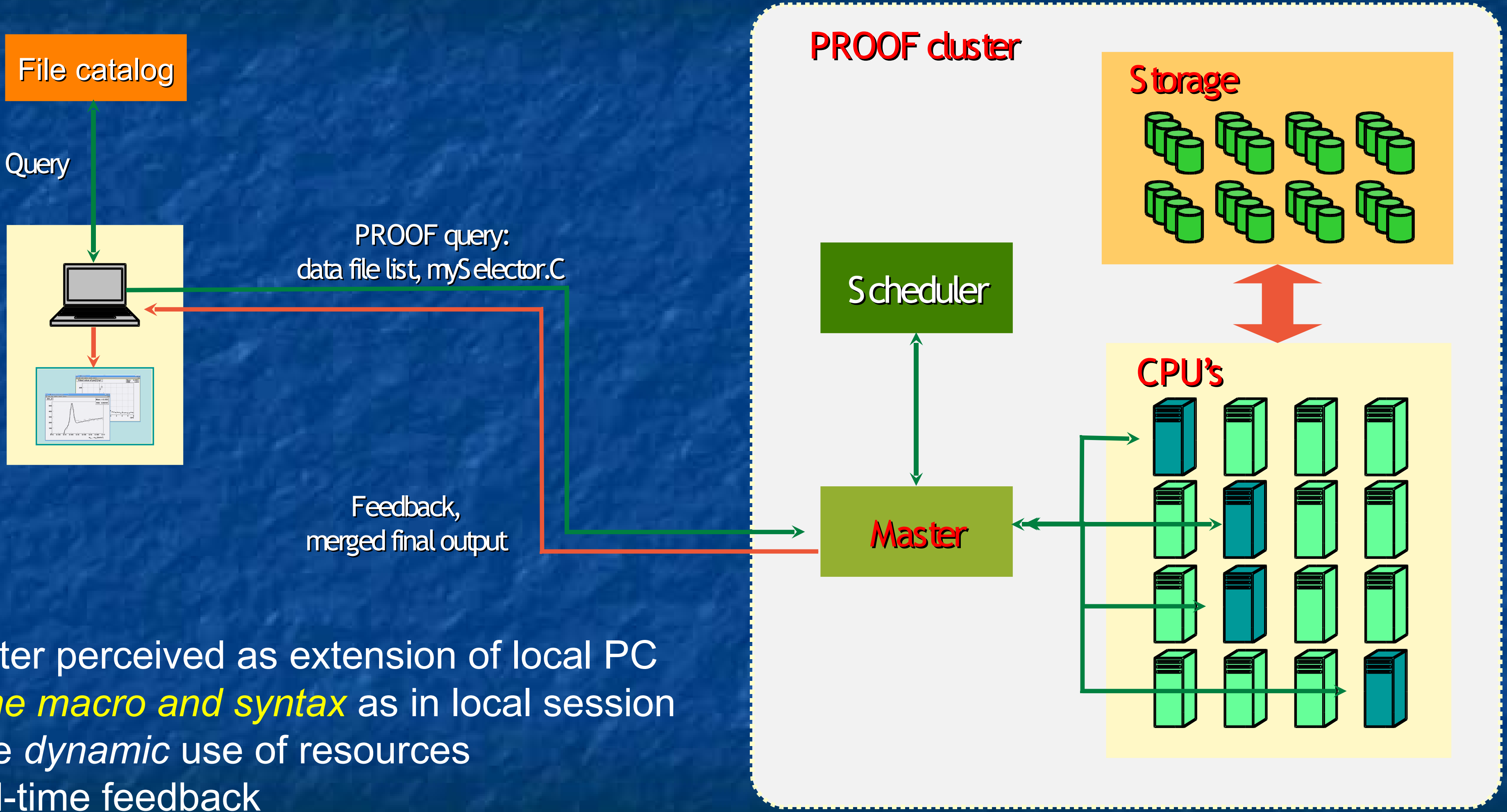
PROOF

- Dynamic approach to end-user HEP analysis on distributed systems exploiting the intrinsic parallelism of HEP data
  - Bring the ROOT model on a distributed cluster
  - Try to process the data where they are
    - Analysis outputs are (typically) small and easy to move
- Transparent, scalable extension of the ROOT shell

Where to use PROOF ?

- Central Analysis Facilities (CAF)
- Department workgroup facilities
- Multi-core, multi-disk desktops

The PROOF Approach



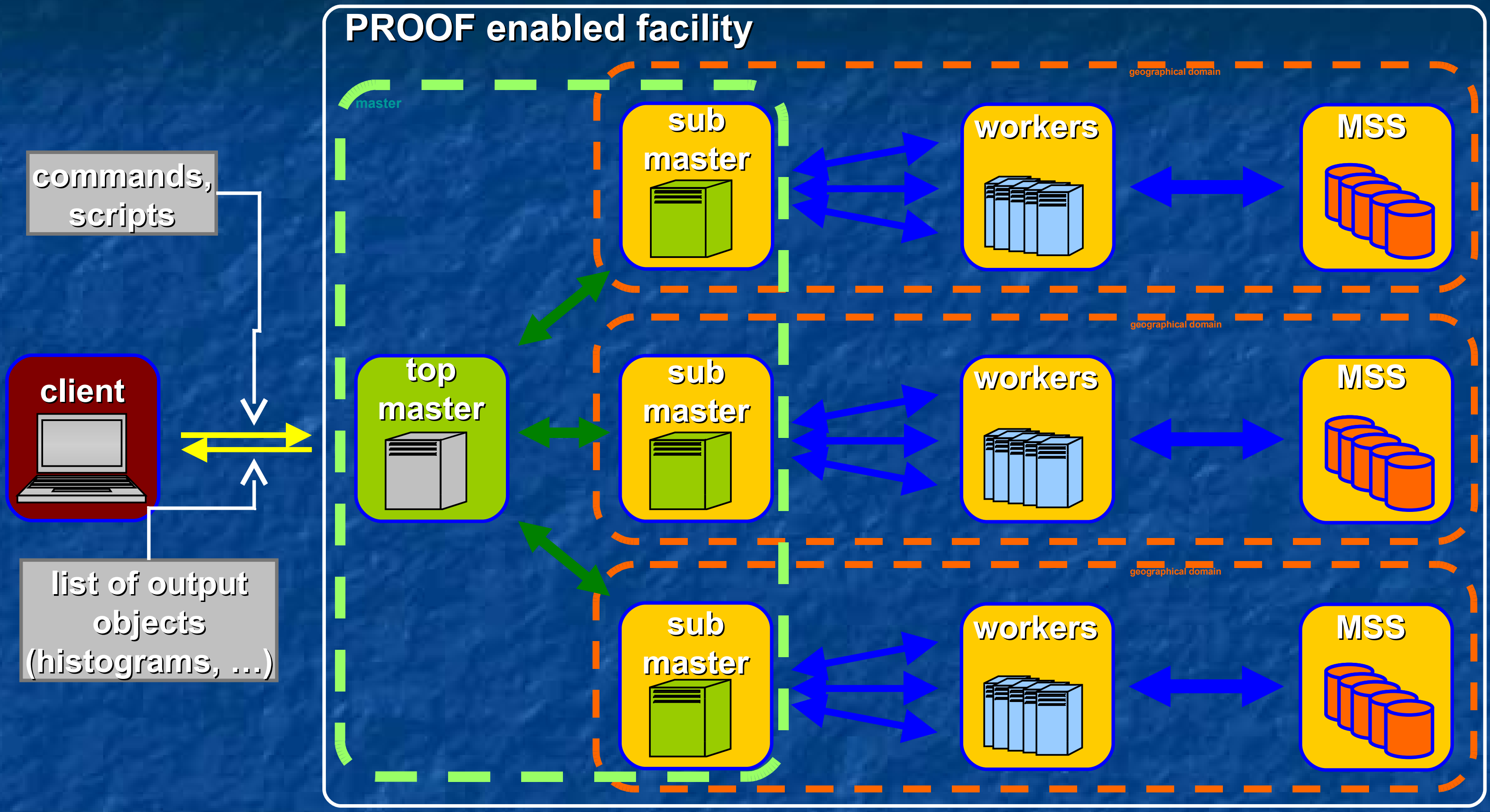
- Cluster perceived as extension of local PC
- *Same macro and syntax* as in local session
- More *dynamic* use of resources
- Real-time feedback
- Automatic *splitting* and *merging*

PROOF architecture

- Three-tier Client-Master-Workers architecture
- Flexible Master tier
  - Adapt to heterogeneous configurations
  - Dilute load of reduction (merging) phase

- Since 2006 connection layout based on SCALLA (xrootd)
  - PROOF daemon is a plug-in to SCALLA (xrootd)
  - Data and PROOF access with the same daemon

PROOF architecture



Network performance: Less important → VERY important

Optimize for data locality or high bandwidth data server access

Interest in PROOF

- Started as joint project MIT (PHOBOS) / CERN (ALICE)
  - Currently CERN / PH-SFT + contributions from GSI
- Used by PHOBOS for end-user analysis since 2003
  - M. Ballantjin et al.
- ALICE pioneered the use at LHC
  - officially required for analysis facilities
  - CAF, GSI
- ATLAS
  - BNL, LMU, Munich, UTA, UA Madrid, Wisconsin
- CMS
  - TWiki page, Analysis framework adapted to PROOF
  - Oviedo (Spain), Purdue
- LHCb started work to adapt pyroot in PROOF (TPySelector)

A bit about PROOF internals

- Simple idea: let the workers to ask for work when idle (pull architecture)
  - Adapt dynamically the work load to their speed
- Unit of work is called packet
  - A range of events inside a file
- The packetizer is the engine that controls all this
    - The heart of the system
    - Workers measured speed is used to determine the size of the next packet
  - Abstract interface: develop / load the most appropriate packetizer strategy

Some PROOF features

- Background running
- Real-time Feedback
  - Define a set of objects sent back at a tunable frequency
- Dataset management (later)
- Package manager
  - Allows to load code needed by the analysis
- Flexible environment setting
  - Can run a script to setup the need vars before the starting the server session (e.g. CMS needs to run SCRAM)

Developments for LHC

- Started Summer 2005 (~2 FTE)
- Move to an XROOTD-based connection layer
  - Stateless connection
  - Coordinator functionality on the Master
- Dataset management
  - Provide basic handling of the local pool storage (xrootd)
- Address multiuser performance issues

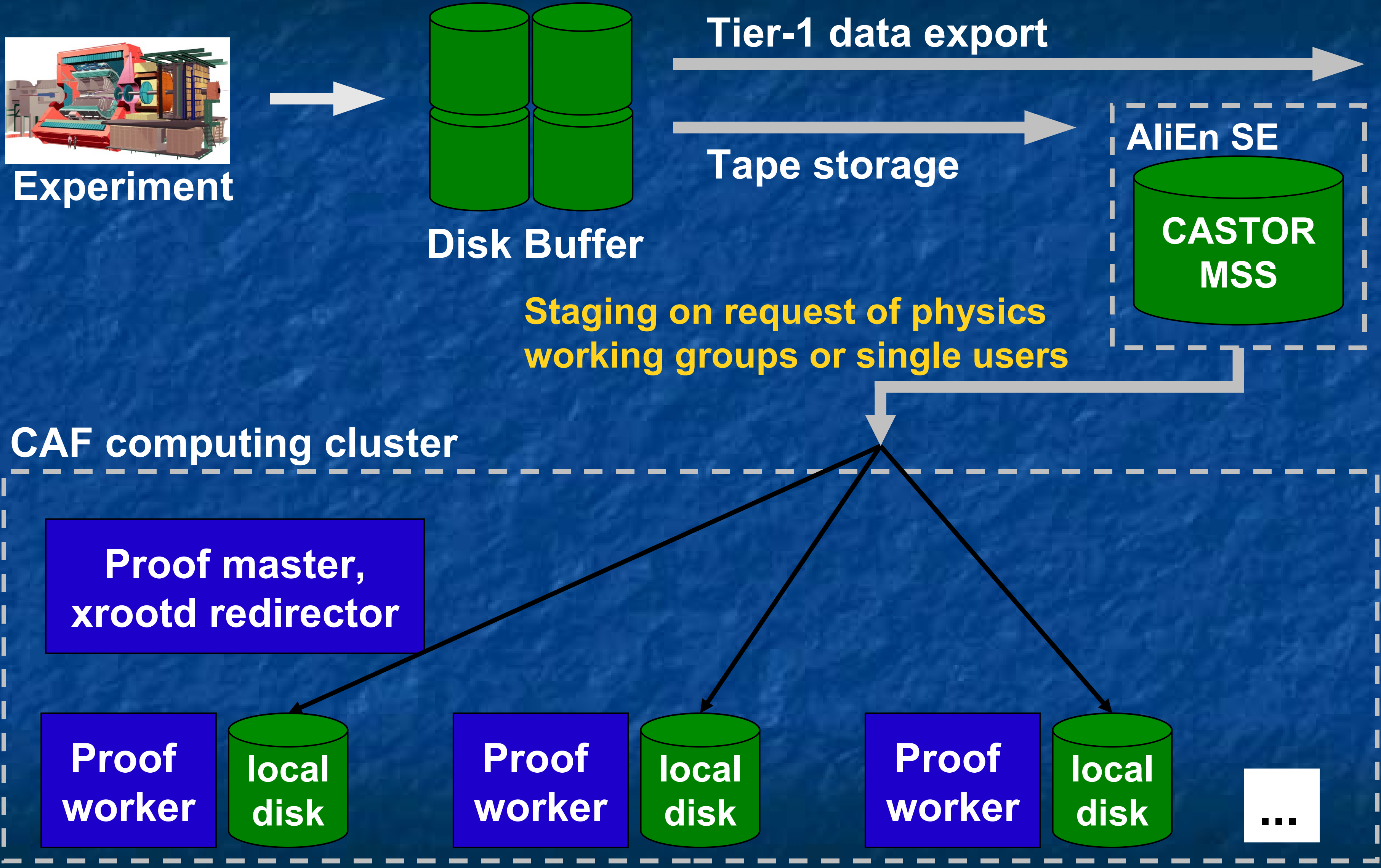
Two main clients contributed to setup the use-cases

- ALICE (spring 2006), ATLAS (spring 2007)

ALICE CAF

- CAF: CERN Analysis Facility
- PROOF enabled cluster for
  - Prompt analysis of pp data
  - Pilot analysis of Pb-Pb data
  - Fast simulation and Reconstruction
  - Calibration & Alignment
- Focus on fast response time
- Design goal: 500 CPUs, 100 TB

ALICE CAF schema



Current ALICE CAF

- CAF testbed
  - 35 dual-core machines, 4 GB RAM, 250 GB disk
    - Additional 5 machines used for development
  - Expected to be enlarged soon to O(100) and more disk
- Tutorials run regurarly to trail people
- 104 users have used at least once the system
- Typically 5 users logged on
  - Mostly analysis framework developers
- Staging data from Alien and Castor
  - About 6-7 TB on disk from recent DCs

Issues addressed by ALICE

- Dataset distribution
- Dataset management
- Enforcing experiment priority policies
- Handling of large outputs
  - Merging from files
  - Attached SE for temporary output files

The dataset manager

- Purpose: provide a tool to easily handle collection of files (datasets), typically identified by name
- Use-cases:
  - A user makes her/his own reduction and ends-up with lists of files on which she/he will run futher, repeated, analyses; these files are typically listed in plain text files
  - Experiment ESD/AOD data runs are conveniently grouped in sets corresponding to similar running conditions; the files are typically taken from a catalog or database
  - Physics groups define the data and Monte Carlo samples adapted to their analysis; the files are typically taken from a catalog or database

TFileInfo, TFileCollection

- TFileInfo is an envelop describing a file in the GRID world
  - One unique identifier (TUUID)
  - Potentiallly many physical / logical locations (list of TUrls)
  - Generic content (list of metadata info, e.g. one per tree contained in the file)
- TFileCollection is a named list of TFileInfo
  - Contains also some meta data about the whole collection, e.g. total entries for a given tree, etc
  - Most generic description of a dataset

TProofDataSetManager

- Master Interface to datasets developed with ALICE
  - RegisterDataSet(const char *name, TFileCollection *fc, …)
    - Creates dataset ‘name’ with files from ‘fc’
  - RemoveDataSet(const char *name, …)
    - Remove the dataset ‘name’
  - ScanDataSet(const char *name, …)
    - Go through dataset ‘name’ and verify that all files are available
  - TFileCollection *GetDataSet(const char *name, …)
    - Get the collection of files composing ‘name’

TProofDataSetManager: quota control

- Basic quota control at group level is provided
- Groups and quotas are defined into a plain text file

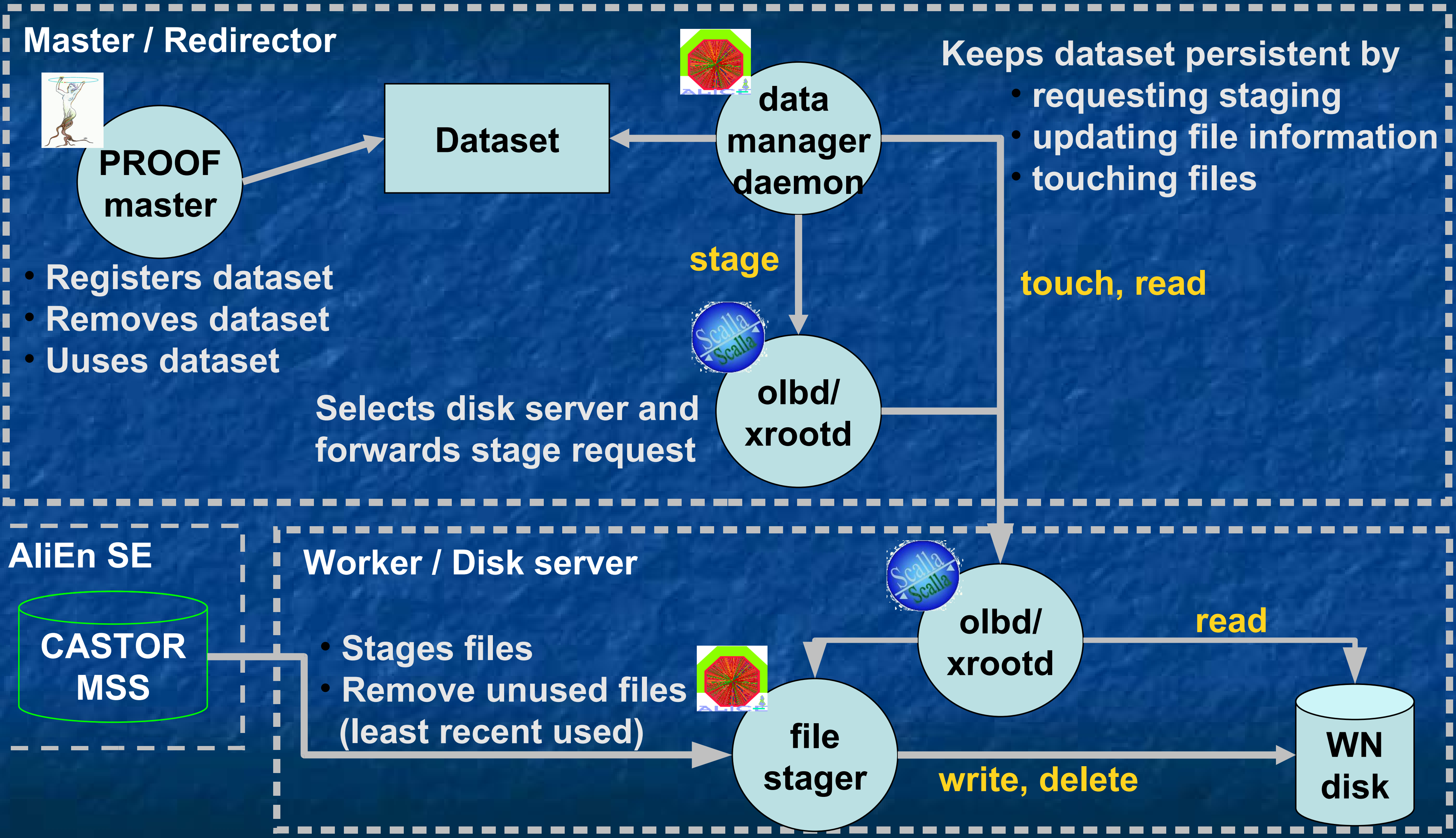
```
# Definition of groups
group PWG1 belikov
group PWG2 jgrosseo, mvala

# Quota control enabled
diskquota on

# Estimated Average file size when real size unknown
averagefilesize 1G

# Quotas
property default diskquota 0
property PWG1 diskquota 1000
property PWG2 diskquota 2000
```

Dataset handling at ALICE CAF



Keeps dataset persistent by

- requesting staging
- updating file information
- touching files

- Registers dataset
- Removes dataset
- Uuses dataset

Selects disk server and forwards stage request

- Stages files
- Remove unused files (least recent used)

TProofDataSetManager: remarks

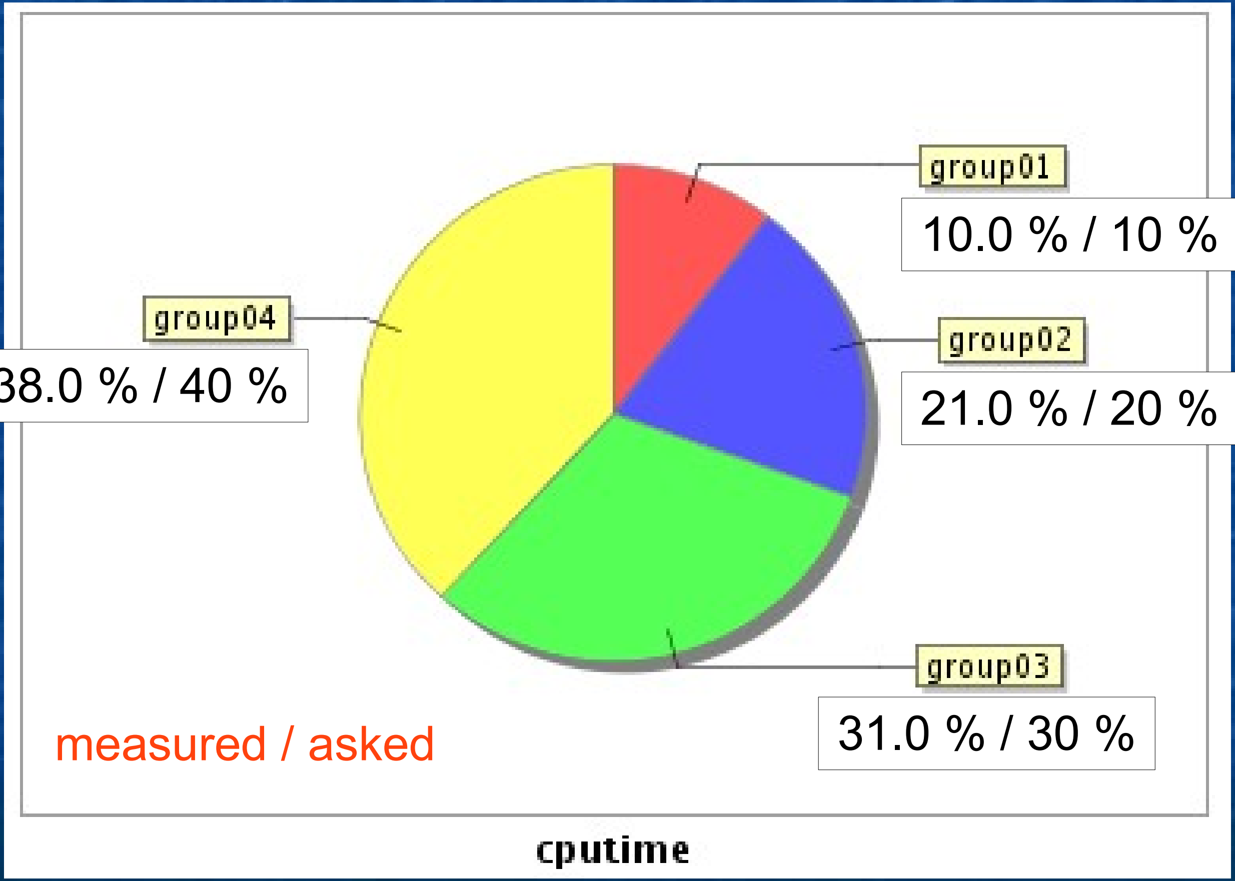
- Not all actions are allowed
  - ALICE does not allow ‘verification’ to avoid overloading the system with too many stage (prepare) requests
  - A dedicated daemon running every 10 min make sure that file movement for newly requested files is started
- ALICE uses as backend the file system
  - TProofDataSetManagerFile handles dataset meta information in ROOT files located under a dedicated directory (/pool/datasets)
  - One file *dsname*.root per dataset
- Development of TProofDataSetManagerSQL started

Enforcing experiment priority policies

- Based on group priority information defined in a dedicated file
- Technology
  - "renice" low priority, non-idle sessions
    - Priority = 20 – nice (-20 <= nice <= 19)
      - Limit max priority to avoid over killing the system
- Can be centrally controlled
  - Master reads updated priorities from the file and broadcast them to the active workers
  - File can be updated by a dedicated process using the info of a monitoring tool
  - Feedback mechanism allows fine-tuning

Example results of renicing technique

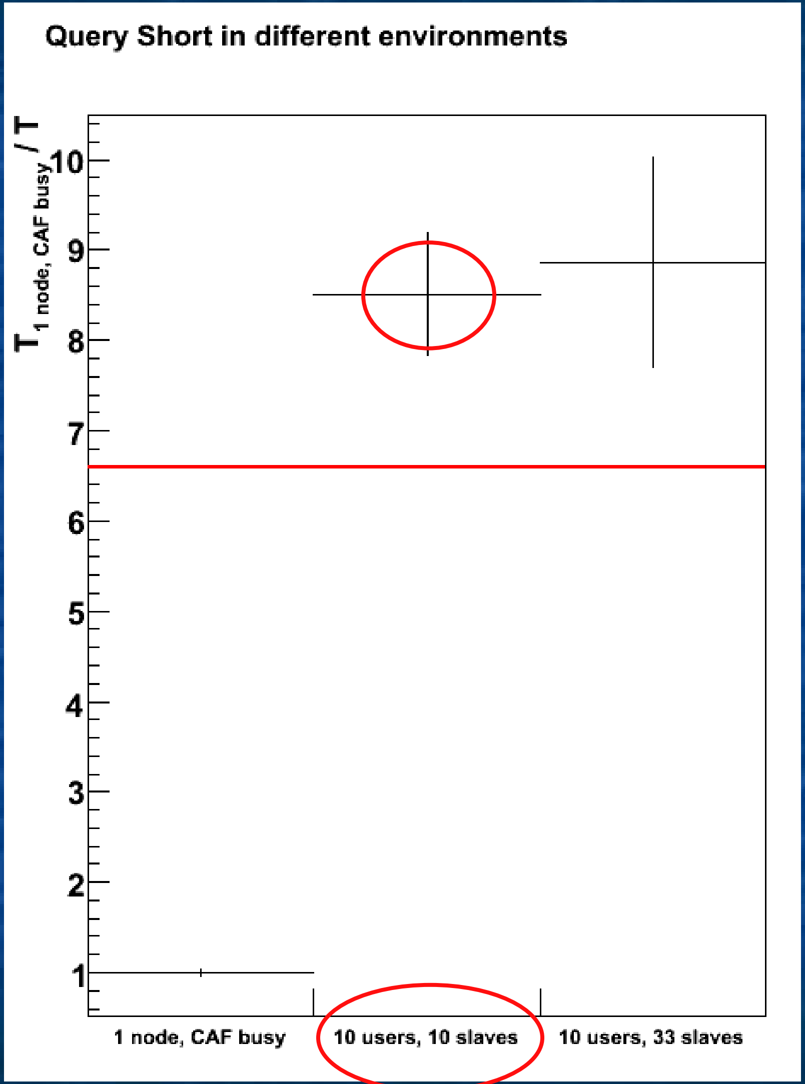
- Stress-test run 4 groups over 1 day

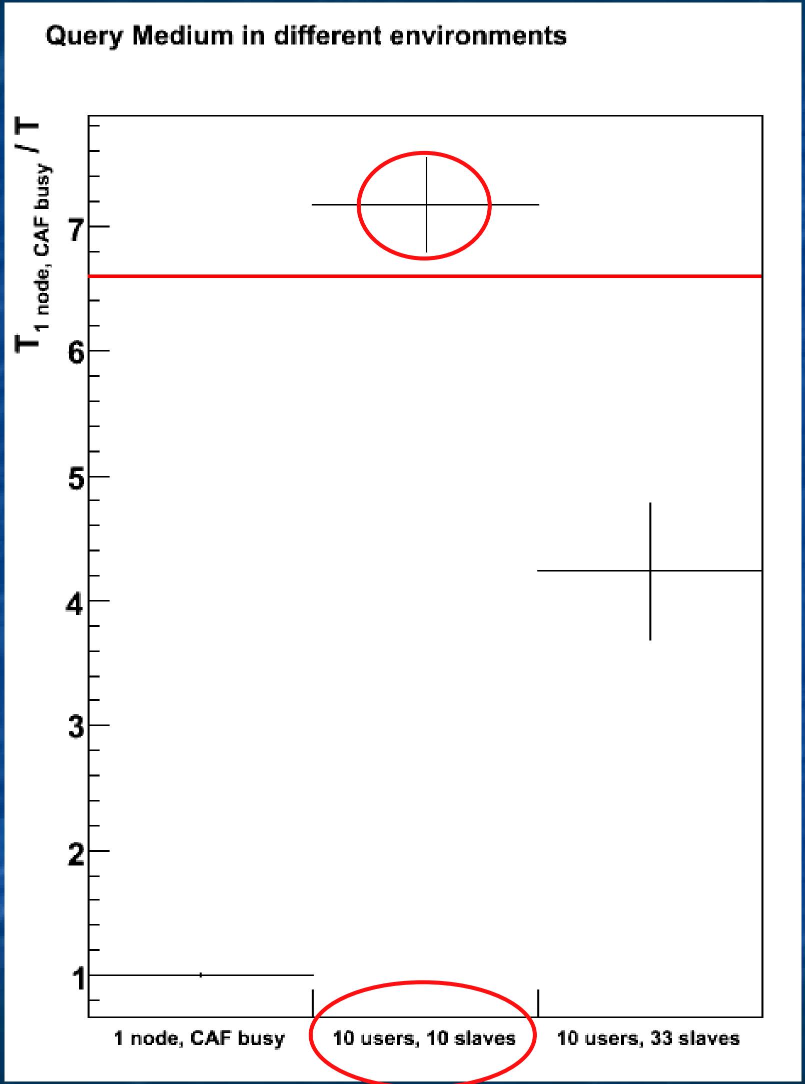


ALICE multiuser experience

- System scheduler does ~well when 5-10 users use cahotically the 34 CAF machines
  - Dont feel them when running cahotic tests
- ALICE run stress tests using cocktails queries submitted contineously with random separation time between queries
  - 4 different query types
    - 20% very short queries (0.4 GB)
    - 40% short queries (8 GB)
    - 20% medium queries (60 GB)
    - 20% long queries (200 GB)
  - 33 nodes available for the test
  - Maximum average speedup for 10 users = 6.6

ALICE multiuser experience (2)





ALICE multiuser experience (3)

- Theoretical batch limit achieved and by-passed automatically
- Machines load was 80-90% during the test
  - Adding workers may be inefficient
- Tune number of workers for optimal response
  - Depends on query type and internals
    - Number, type, size of output objects
  - Shows importance of active scheduling

Central Scheduling

- Control resources and how they are used
- Improving efficiency
  - assigning to a job those nodes that have data which needs to be analyzed.
- Implementing different scheduling policies
  - e.g. fair share, group priorities & quotas
- Efficient use even in case of congestion

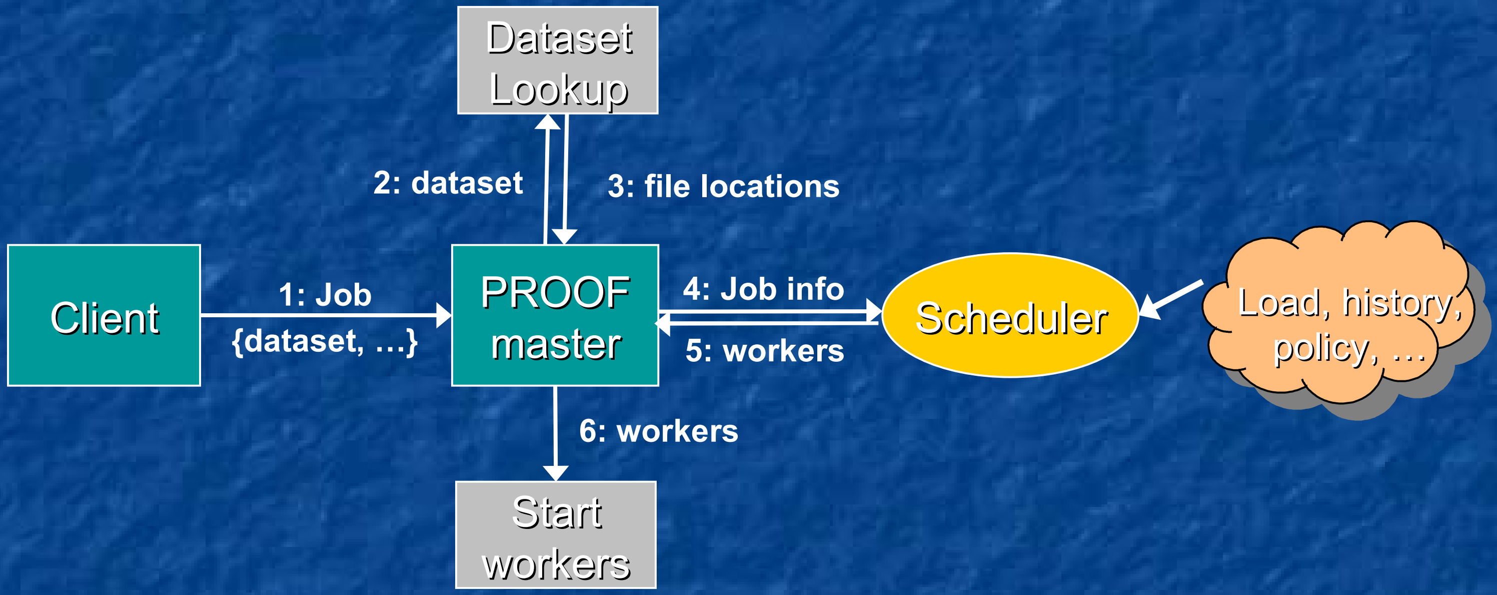
Central Scheduling: on-going work

- Assigning a set of workers for a job based on:
  - The data set location
  - User priority (Quota + historical usage)
  - The current load of the cluster
- Create (priority) queues for queries that cannot be started

Central Scheduling: status

- Implementation exists with:
  - # of Workers ≈ relativePriority * nFreeCPUs
  - Assign least loaded workers first
- Missing pieces
  - Dynamic worker setup
  - Worker nodes auto-registration
    - Improved load monitoring
  - Support for “put-on-hold” submission
    - Prototype on test

Central Scheduling: schema



Merging via files

- Large output objects (e.g. trees) create memory problems
  - E.g. ALICE wants to create sample AODs at CAF
- Solution:
  - save them in files on the workers
  - merge the files on the master using TFileMerger
- New class TProofOutputFile defines the file and provide tools to handle the merging
  - Unique file names are created internally to avoid crashes
- Merging will happen on the Master at the end of the query
- Final file is left in sandbox on the master or saved where the client wishes
  - E.g. ALICE wants to register them to AliEn

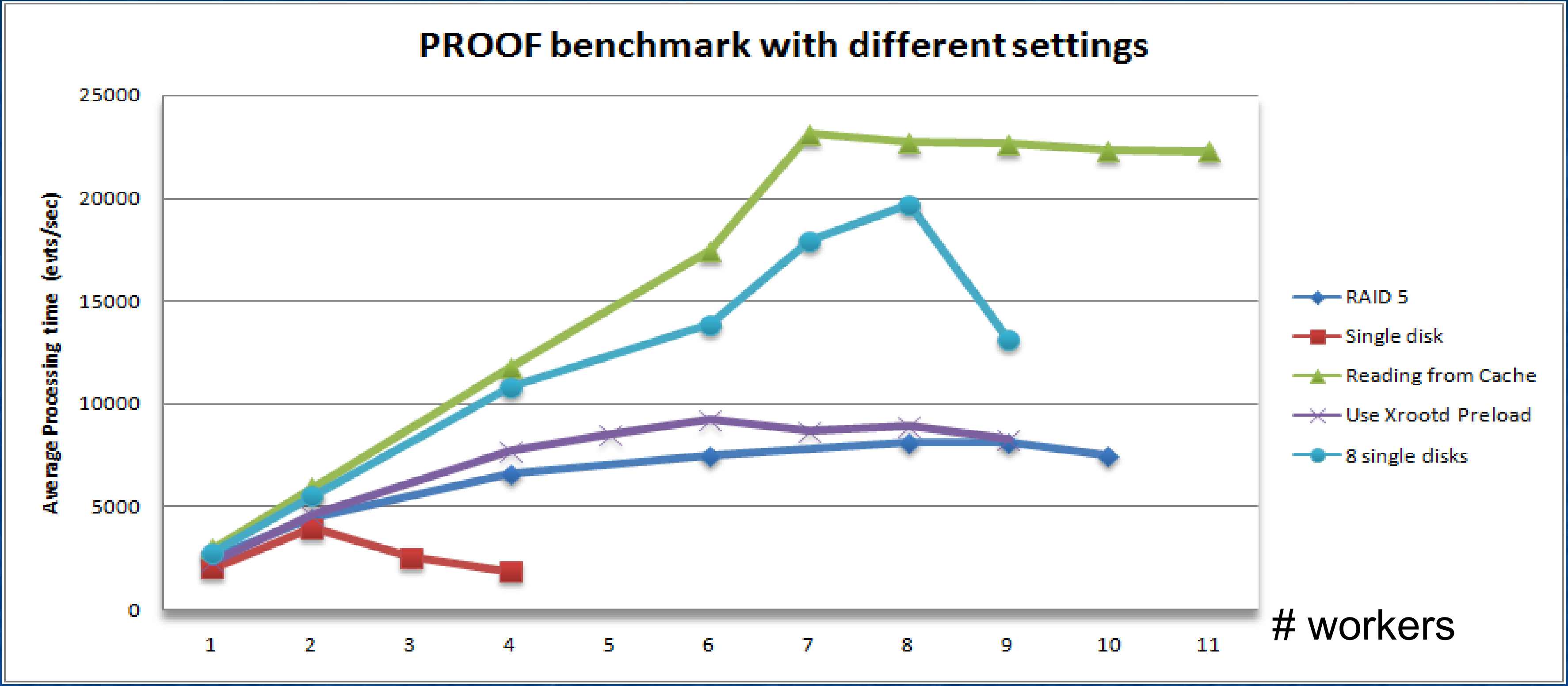
Issues addressed by ATLAS

- Performance issues
- Federation
  - Dataset issues
- Condor integration

Performance studies

- Nice work being done by Neng Xu (UW) and Sergey
- Summarized by them at PROOF07
- Next slides show summary of Neng's results
  - Hardware
    - Intel 8 core, 2.66GHz, 16GB DDR2 memory.
      - With single disk.
      - With 8 disks without RAID
      - With RAID 5(8 disks)
  - PROOF benchmark code

Performance studies (2)



Courtesy of Neng Xu

- RAID scales well but single disks are better
  - trade-off between the data protection and the performance
- 2 cores vs. 1 disk seems to be a reasonable HW ratio without special technologies

Considerations about I/O HW

- ROOT can handle ~10 MB/s
- Rate is given by

$$\frac{1}{R} = \left(\frac{1}{R_d(x)} + \frac{1}{R_{I/O}} \right) \cdot x$$

- Where x is the compression factor (~2), $R_d(x)$ is the decompressing rate, and $R_{I/O}$ is the I/O rate
- For $R_{I/O}$ ~ 40 MB/s and x ~ 2, $R_d(x)$ is ~ $R_{I/O}$
  - Single core I/O reduces effective I/O by 2

Considerations about I/O HW (2)

- With multicore you could think of dedicate one or more cores to decompression
- Rate is then given by

$$\frac{1}{R}=Max\left(\frac{1}{R_d(x)},\frac{1}{R_{I/O}}\right)\cdot x$$

- You can get the real I/O speed
- Undergoing development to do decompression in a separated thread

Dataset issues in federated cluster

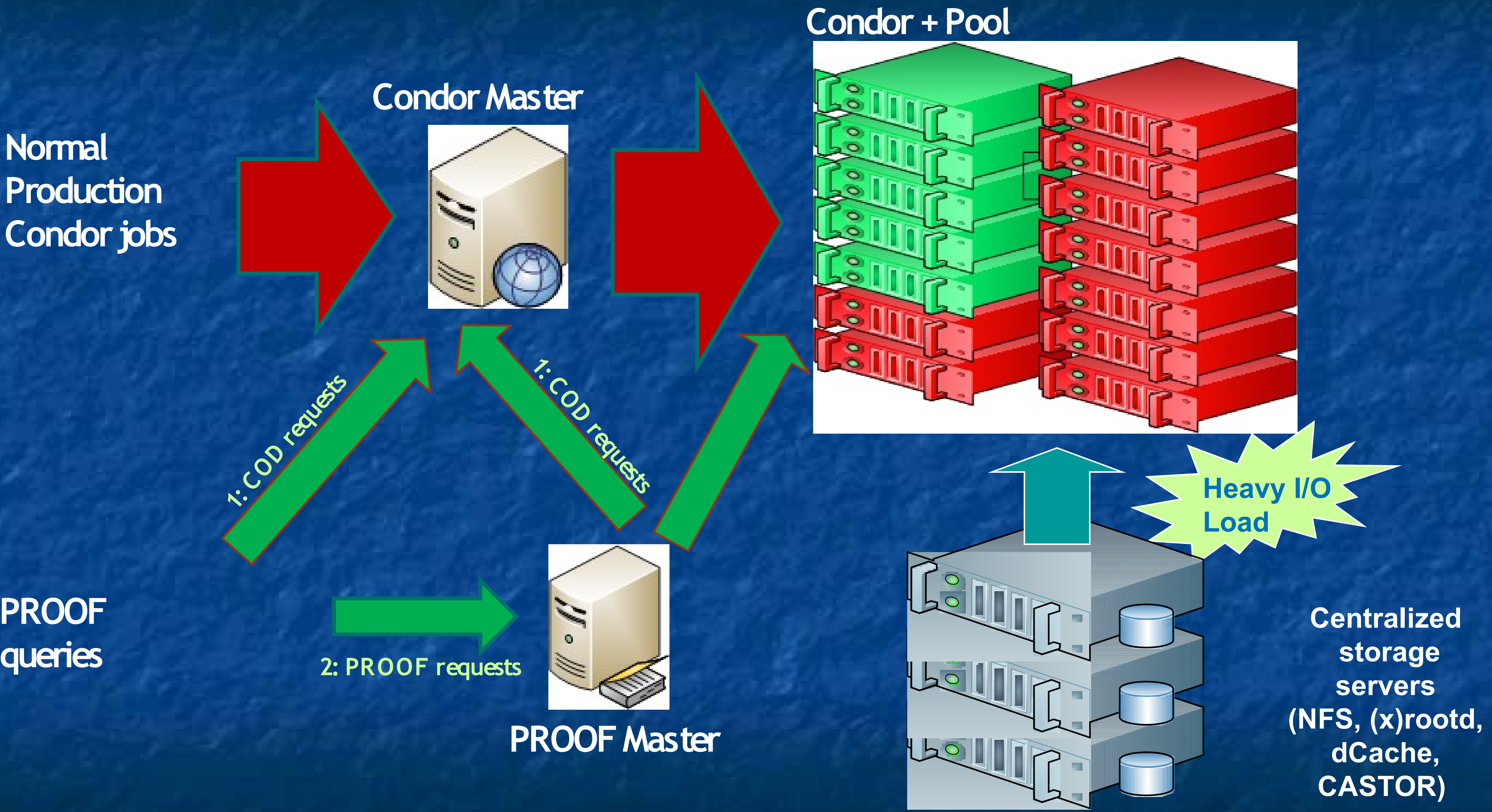
- Multi-Master level allows to put together cluster from geographially separated domains
- However dataset information is local to each sub-cluster
- Needs to make the top-master (entry point) aware of the location of the requested dataset
- Or to have a way for fast querying

Condor integration

- Neng Xu et al., Wisconsin
- Improved integration at master coordinator level
- Condor has full control of processes
  - PROOF servers, file movers, xrootd

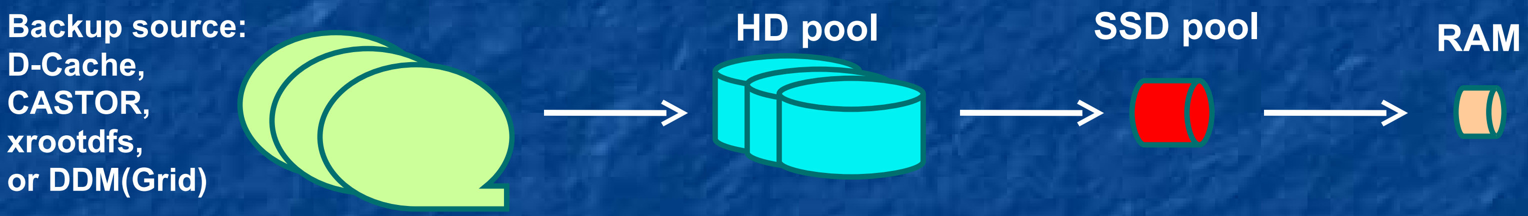
- PROOF required development
  - Support for query “put-on-hold” / preemption
    - Prototype existing
  - Support for server startup via Condor

Condor Integration: first model using COD



The new model (N. Xu et a. Wisconsin)

- Designed for a large scale Analysis Facility with
  - PROOF pool
  - > 100 users
  - Limited and structured storage



- Issues:
  - Efficient scheduling of large number of users
  - Data staging-in/-out on the pool

The new model: flow

- Users issue PROOF processing queries in normal way
- PROOF master
  - Puts the query on-hold
  - Creates a Condor job for each query
    - Dataset readiness as requirement ClassAd
  - Submits the job to the Condor scheduler
- Condor scheduler puts the job in Held state
- File stager daemon checks regurarly for new dataset requests
  - Collects the dataset requirements and arranges the movement of files
  - Releases the processing job when their required datasets are ready
- Condor scheduler runs the job on PROOF, resuming the query put on-hold at the previous step

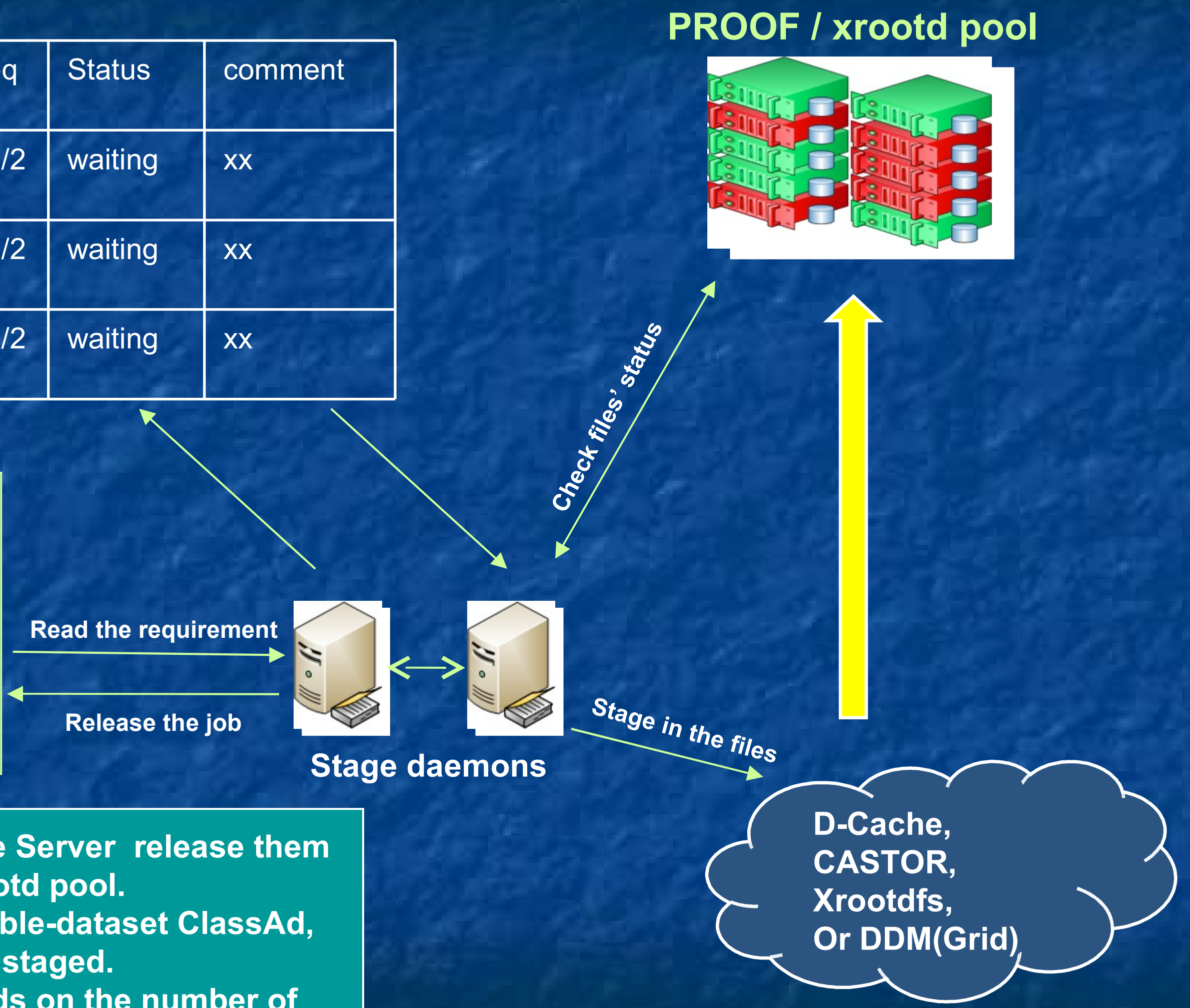
The new model: file staging

Database for Datasets

Dataset name	#of req	# of file	Last req date	Status	comment
mc08.017506.PythiaB_bb mu6mu4X.evgen.e306	2	50	2008/2/2 0	waiting	xx
mc08.017506.PythiaB_bb mu6mu4X.evgen.e306	1	50	2008/2/2 0	waiting	xx
mc08.017506.PythiaB_bb mu6mu4X.evgen.e306	1	500	2008/2/2 0	waiting	xx

PROOF batch jobs list

Name	input dataset	
Job1	mc08.017506.PythiaB_bbmu6mu4X.evgen.e306	500
Job2	mc08.017506.PythiaB_bbmu6mu6X.evgen.e306	400
Job3	mc08.0175068.PythiaB_bbmu6mu4X.evgen.e306	30
Job4	mc08.017506.PythiaB_bbmu6mu4X.evgen	50
Job5	mc08.017888.PythiaB_bbmu6mu4X.evgen.e306	100
Job6	mc08.017506.PythiaB_bbmu6mu4X.evgen.e306	120



- Condor jobs set to "Held" by default. The Stage Server release them once the dataset is staged into the PROOF / Xrootd pool.
- Or Condor jobs declares dependency on available-dataset ClassAd, updated by the Stage Server once the dataset is staged.
- Dataset stage-in also has priority which depends on the number of requests, number of files, waiting time, etc..

The new model: node configuration

Condor Scheduler for PROOF

Master

Service for Scheduling
Condor Master
Condor Collector
Condor Scheduler

These slots can be used to limit the total number of running PROOF sessions

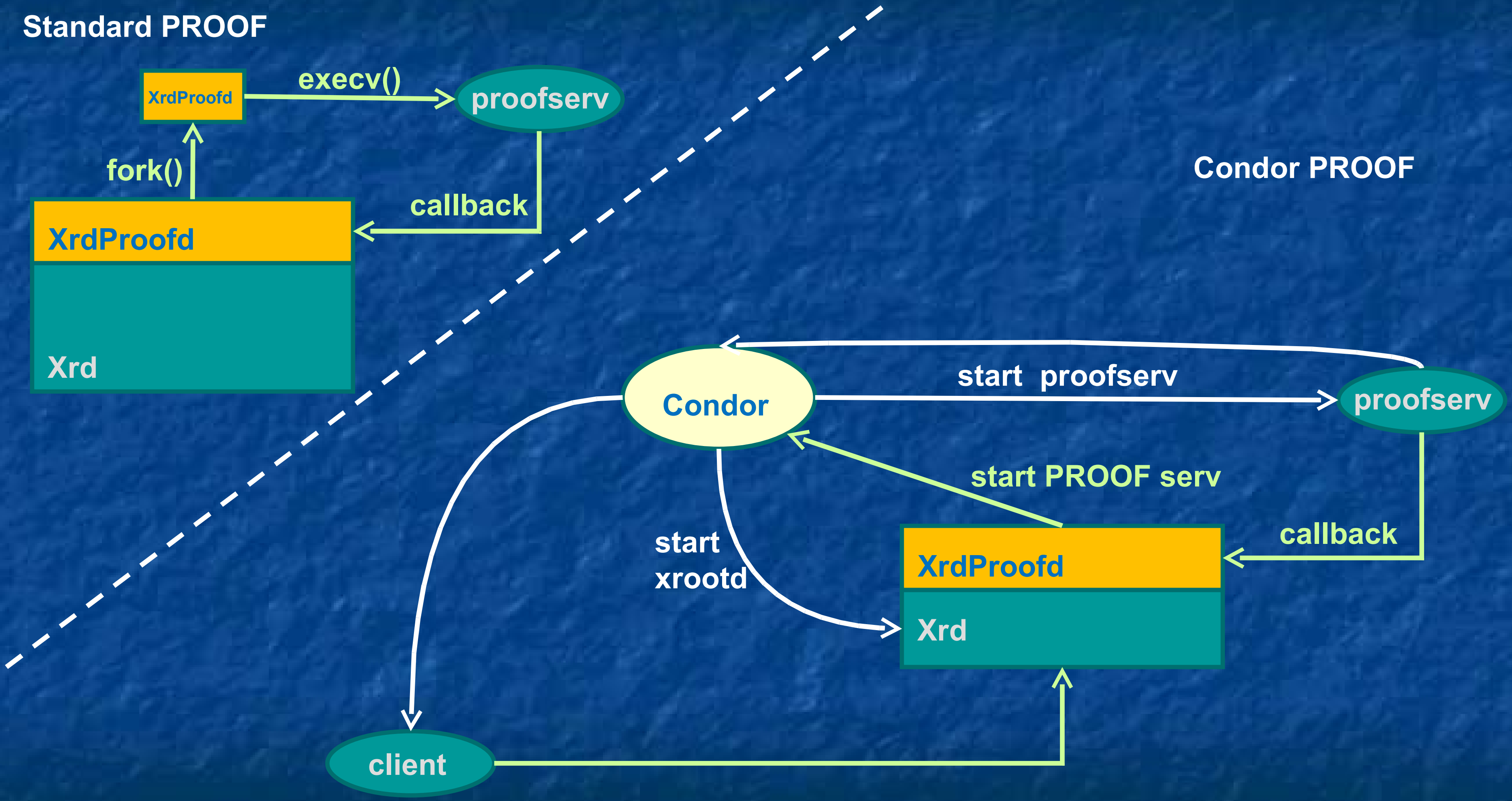
Worker

Service for PROOF jobs
Condor Starter

Job slots for PROOF session
slot1@pcuw104
slot2@pcuw104
slot3@pcuw104
slot4@pcuw104

Job slots for File Stage-In (can run on background)
slot5@pcuw104
slot6@pcuw104
slot7@pcuw104
slot8@pcuw104
slot9@pcuw104
slot10@pcuw104

PROOF Condor interaction



Other developments

- New version of XrdProofd
- PROOF-Lite
- Real-time memory monitoring
  - Report regurarly memory comsumption in the dialog
  - HWM and stop-processing threasholds
- Admin area sharing between master
  - Based on distributed file system
  - Embedded using xrootd
  - Need for master redundancy

New XrdProofd

- Improved stability
  - Fixes possible deadlocks following wild usage of TProof::Reset()
- Resiliance to XROOTD failure
  - Attempts to reconnect for 10 mins
- Framework for node auto-registration

PROOF Lite

- PROOF Lite is a realization of PROOF in 2 tiers
  - The client starts and controls directly the workers
  - Communication goes via UNIX sockets
- No need of daemons:
  - workers are started via a call to 'system' and call back the client to establish the connection
- Starts N_{CPU} workers by default

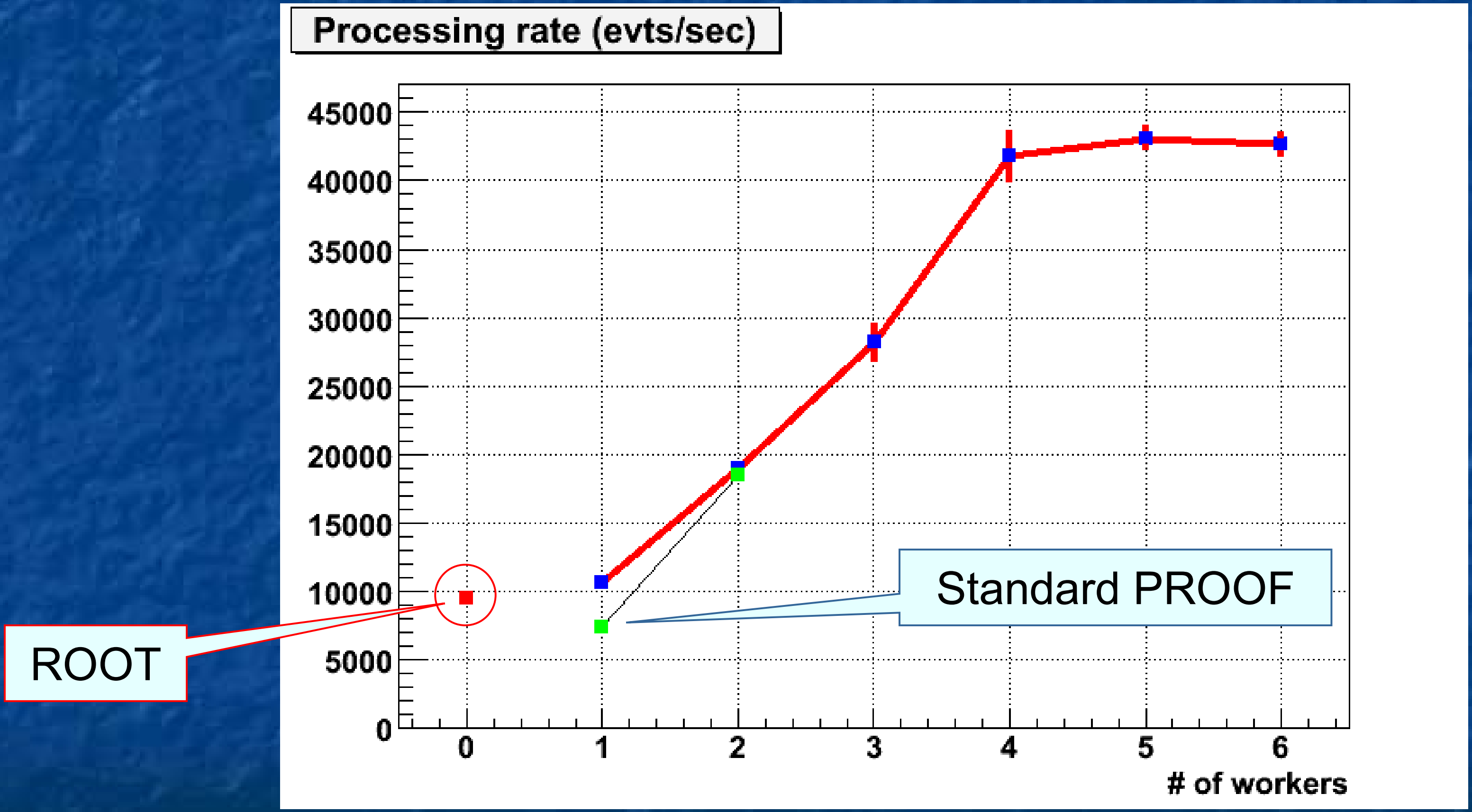
PROOF Lite (2)

Additional reasons for PROOF-Lite

- Can ported on Windows
  - There is no plan to port current daemons to Windows
  - Needs a substitute for UNIX sockets
    - Use TCP initially
- Can be easily used to test PROOF code locally before submitting to a standard cluster
  - Some problems with users’ code are difficult to debug directly on the cluster

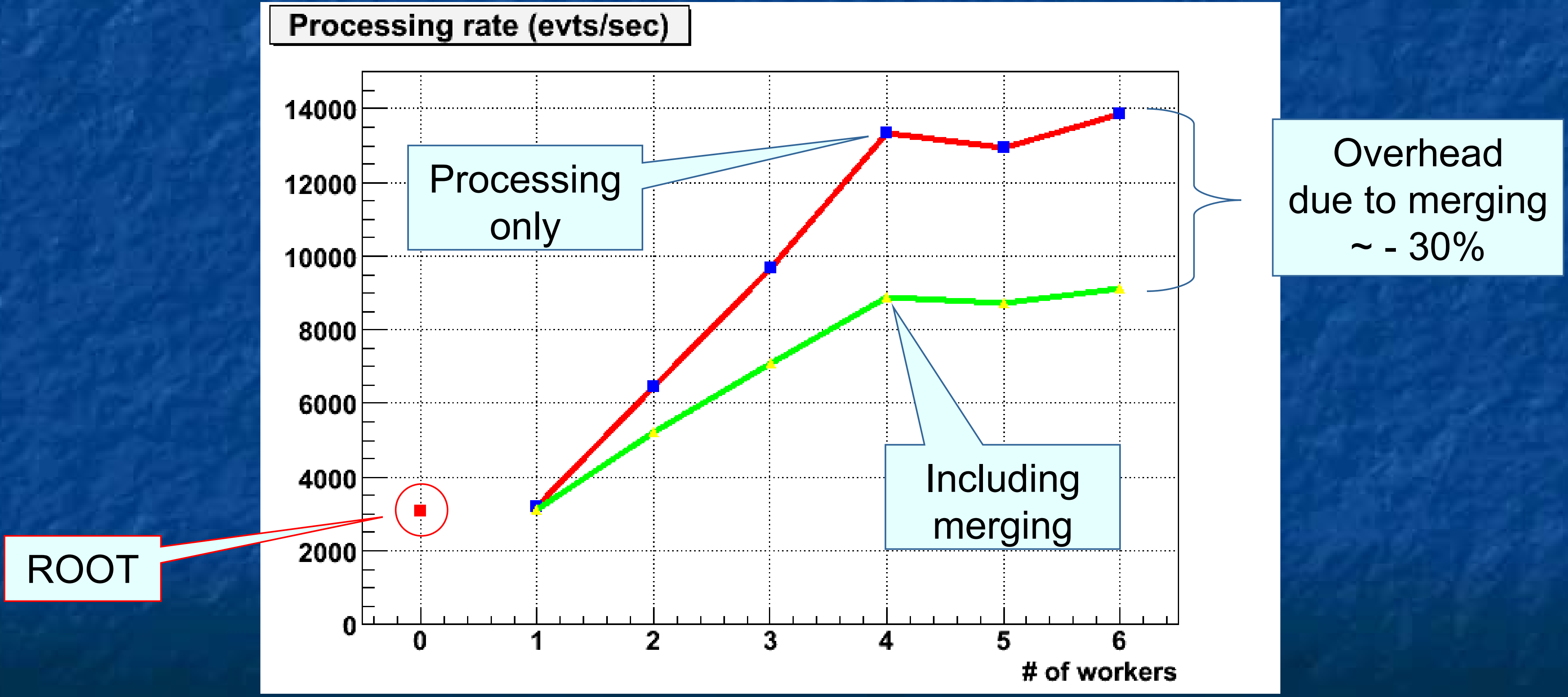
Simple scaling

- Simple event generation and 1D histogram filling



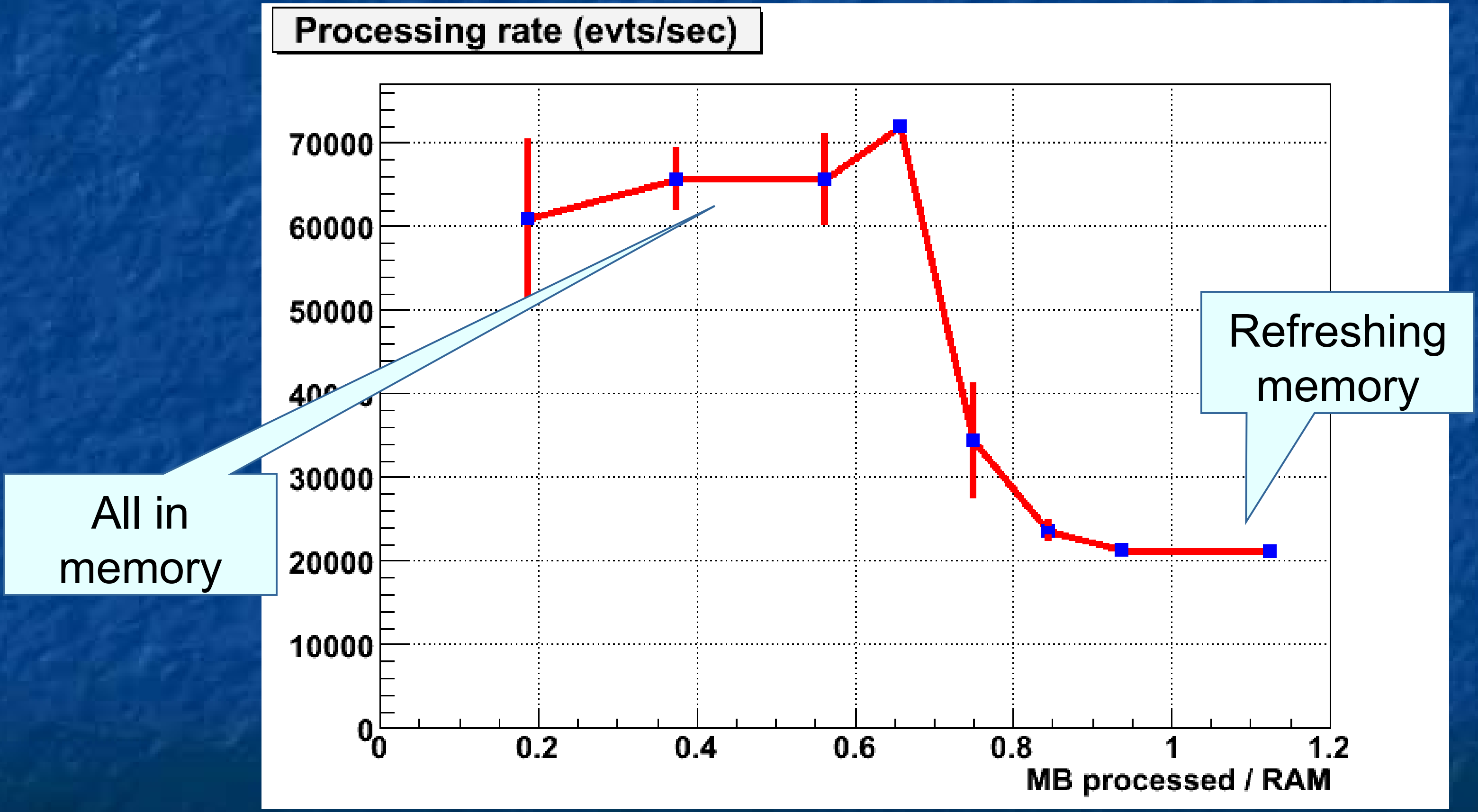
Simple scaling ~large output

- Simple event generation, TTree filling, merging via file (output ~ 350MB after compression)



Performance vs fraction of RAM

- Reading datasets of increasing size



Summary

- Many new developments in view of LHC data
- Interest PROOF is growing steadly
- Lot of activities going on on PROOF trying to meet experiments needs
  - Experiment feedback essential and very valuable
- The system is getting ready for the real data

Thanks to

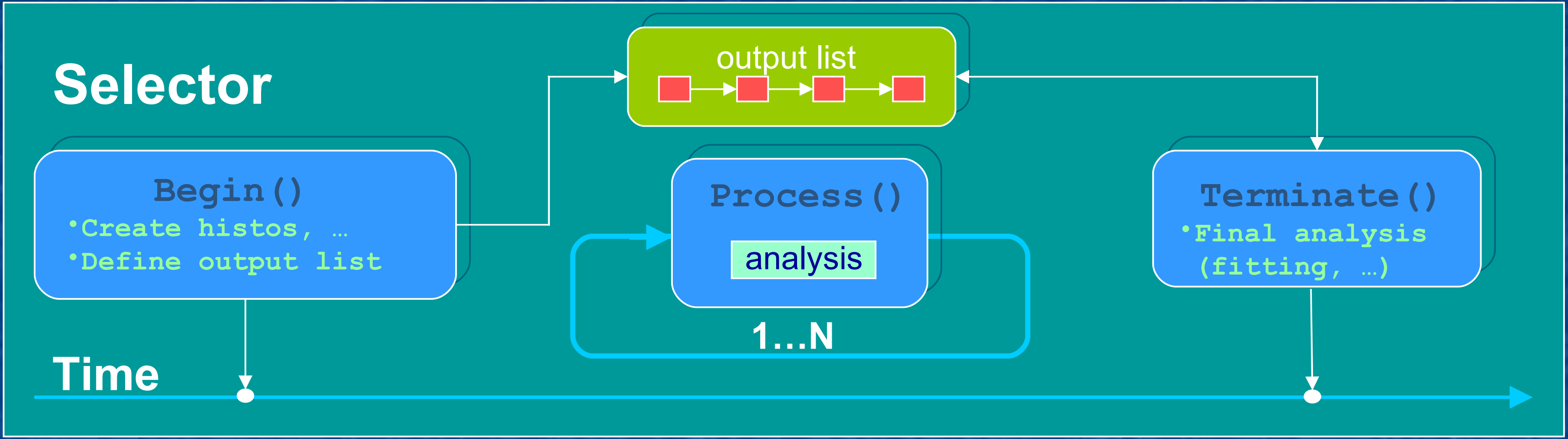
- CERN / PH-SFT: J. Iwaszkievicz, F. Rademakers
- ALICE: J.F. Oetringhaus
- Wisconsin: Neng Xu, M. Livny
- ATLAS: S. Panitkin
- …

Backup

- Non-data driven processing

Generic, non-data-driven analysis

Implement algorithm in a TSelector



New TProof::Process(const char *selector, Long64_t times)

```
// Open the PROOF session
root[0] TProof *p = TProof::Open("master")
// Run 1000 times the analysis defined in the
// MonteCarlo.C TSelector
root[1] p->Process("MonteCarlo.C+", 1000)
```